蒋超，PA2

一、

1. 当， 有解且唯一，r为A、B的秩，n为维度。

2. 高斯消元法，也就是我们中学学过的消元法，将方程组中一方程的未知数用含有另一未知数的代数式表示，并将其带入到另一方程中，这样就消去了一个元，得到一个解。高斯消元法主要用于二元一次方程组的求解。

3. QR分解（正交三角）

定义： 对于n阶方阵A，若存在正交矩阵Q和上三角矩阵R，使得A = QR，则该式称为矩阵A的完全QR分解或正交三角分解。

过程：其中，正交矩阵Q来自Gram-Schmidt正交化方法，R来自。

作用：是目前求一般矩阵全部**特征值和特征向量**最有效且广泛应用的方法。

4. Cholesky 分解（平方根法）

定理：若 对称正定，则存在一个对角元为正数的下三角矩阵，使得成立。

过程：即利用该等式，开方、递推计算。

作用：当线性方程组 ，其中A为正定对称矩阵时，有特定解法。

5.

代码：

#include <iostream>

using namespace std;

#include <ctime>

#include <Eigen/Core>

#include <Eigen/Dense>

#define MATRIX\_SIZE 100

int main() {

// 如果不确定矩阵大小，可以使用动态大小的矩阵

Eigen::Matrix< double, Eigen::Dynamic, Eigen::Dynamic > matrix\_NN;

// 我们求解 matrix\_NN \* x = v\_Nd 这个方程

// Eigen::Matrix< double, MATRIX\_SIZE, MATRIX\_SIZE > matrix\_NN;

matrix\_NN = Eigen::MatrixXd::Random( MATRIX\_SIZE, MATRIX\_SIZE );

Eigen::Matrix< double, MATRIX\_SIZE, 1> v\_Nd;

v\_Nd = Eigen::MatrixXd::Random( MATRIX\_SIZE,1 );

clock\_t time\_stt = clock(); // 计时

// 直接求逆

Eigen::Matrix<double,MATRIX\_SIZE,1> x = matrix\_NN.inverse()\*v\_Nd;

cout<<"time use in normal inverse is " << 1000\* (clock() - time\_stt)/(double)CLOCKS\_PER\_SEC << "ms"<< endl;

// 通常用矩阵分解来求，例如QR分解，速度会快很多

time\_stt = clock();

x = matrix\_NN.colPivHouseholderQr().solve(v\_Nd);

cout <<"time use in Qr decomposition is " <<1000\* (clock() - time\_stt)/(double)CLOCKS\_PER\_SEC <<"ms" << endl;

// Cholesky 分解

time\_stt = clock();

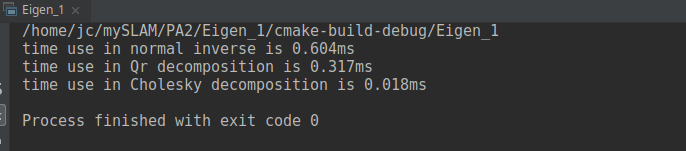
x = matrix\_NN.llt().solve(v\_Nd);

cout<<"time use in Cholesky decomposition is "<<1000\* (clock() - time\_stt)/(double)CLOCKS\_PER\_SEC <<"ms" << endl;

return 0;

}

*运行结果：*



二、

代码：

#include <iostream>

#include <Eigen/Core>

#include <Eigen/Geometry>

int main() {

Eigen::Quaterniond q1(0.55, 0.3, 0.2, 0.2);

Eigen::Vector3d t1(0.7, 1.1, 0.2);

Eigen::Quaterniond q2(-0.1, 0.3, -0.7, 0.2);

Eigen::Vector3d t2(-0.1, 0.4, 0.8);

Eigen::Vector3d p1 = Eigen::Vector3d(0.5, -0.1, 0.2);

Eigen::Vector3d p;

Eigen::Vector3d p2;

// Tcw1

Eigen::Isometry3d Tcw1 = Eigen::Isometry3d::Identity();

Tcw1.rotate(q1.normalized());

Tcw1.pretranslate(t1);

// Tcw \* p = p1

p = Tcw1.inverse() \* p1;

std::cout<<"p = "<<std::endl;

std::cout<<p.transpose()<<std::endl;

// Tcw2

Eigen::Isometry3d Tcw2 = Eigen::Isometry3d::Identity();

Tcw2.rotate(q2.normalized());

Tcw2.pretranslate(t2);

// Tcw2 \* p = p2

p2 = Tcw2 \* p;

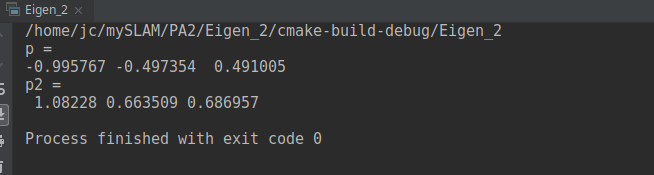
std::cout<<"p2 = "<<std::endl;

std::cout<<p2.transpose()<<std::endl;

return 0;

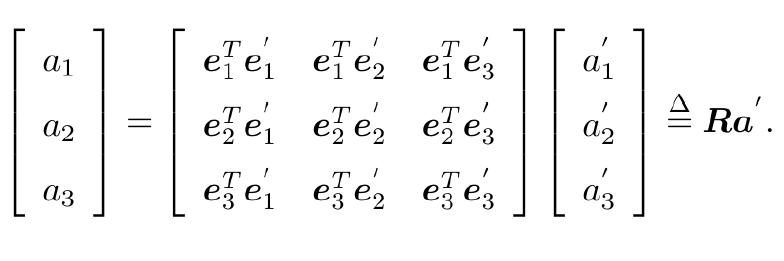
}

运行结果：



三、

1. 由



中，矩阵可改写为，即，可以的出; 即

2. 维度为3，η维度为1

3. 设: , , , ，, , ,

则：

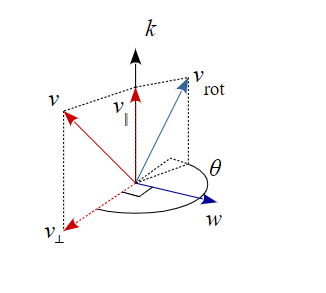
根据矩阵乘法规则与四元数运算法则，可以计算出该结果等于。

同理:

本题意义在与将四元数乘法转化为矩阵乘法。

有疑问： 中是什么形式？ 是矩阵还是四元数？难道可以推出，是 否正确？？

四、



证明：

先考虑对一个向量作旋转，其中是原向量，三维的单位向量是旋转轴，是旋转角度，是旋转后的向量。

先通过点积得到v在k方向的平行分量。

在通过叉乘得到与k正交的两个向量和。

······⑴

这样就得到了3个相互正交的向量，如图所示，根据叉乘定义可以得出（此处为难点）：

= ()

继而得出：

······⑵

再引入**叉积矩阵**的概念；记K为 的叉积矩阵，则K为一个反对称矩阵：

其有如下性质：

为了利用该性质，需要将替换为与的叉积关系，先根据⑴式做代换：

······⑶

根据⑴⑵⑶得到：

根据叉积矩阵的性质：

假设原坐标系基向量矩阵为B，旋转后的坐标系基向量矩阵为C，那么：

将替换为B、C：

同时右乘，即为罗德里格斯公式的标准形式。

证明完毕。

感想：并不复杂，但要清楚理解**叉积及叉积矩阵**的含义。

五、

1.根据第4提结果可得提示中的(6)式，设

则 = (s, -x, -y, -z) /

经计算 的第一行为0，即实部为0

2. 由(6)式易得：

六、

1.代码段17行，使用了范围for循环，实现了类似foreach功能，对vector容器中每一个A对象a，都打印了a.index。

2.代码段17行，使用了自动类型推导关键词auto，编译器会根据上下文情况，确定auto变量的真正类型，此处类型为A。

3.代码段16行，sort函数第三个参数为lambda表达式，这里定义了可调用对象，接受两个A类型变量的常量引用，返回是否第一个的index属性值小于第二个，由于函数体只有一个return语句且没有指定返回类型，表达式将自动推断返回类型为bool。

4.代码段15行，使用了花括号对容器对象进行列表初始化，更加优雅。

5.代码段16行，使用了vector对象的容器迭代器begin()和end()，用来遍历所有元素。